

SECURING STACK OVERFLOW VULNERABILITIES BY MEMORY ACCESS VIRTUALIZATION AND AUTOMATIC PATCHES

LALWANI JAI

Alamuri Ratnamala Institute of Engineering and Technology, Shahapur, Thane, Maharashtra, India

ABSTRACT

System security and availability are the factors seriously affected due to memory vulnerabilities. Buffer overflow attacks still pose a significant threat to the security and availability of today's computer systems. Our proposed system is to provide adequate protection against buffer overflow attacks as most of existing solutions terminate the vulnerable program when the buffer overflow occurs, effectively rendering the program unavailable. Our System, SafeStack, can automatically diagnose and patch stack-based buffer overflow vulnerabilities. The basic idea is to virtualize memory access and move the vulnerable buffer into protected memory regions, which provides a fundamental and effective protection against recurrence of the same attack without stopping normal system execution.

KEYWORDS: Securing Stack Overflow Vulnerabilities, Memory Access Virtualization, Automatic Patches, SafeStack

INTRODUCTION

Computer world is growing fast but it needs system security and availability. Memory vulnerabilities is the biggest threat for system. In network services hijack attacks are found mostly and its need to control it which leads to vulnerabilities. Hijack attacks are detected and also prevented, which is not sufficient for systems. To avail system and network security it is not sufficient to repair the attack, it should be prevented. To control programs attackers exploits memory vulnerabilities. Buffer overflow vulnerabilities is the biggest threat to system and network security. If user is still prone to continue using same software it may cause various problems for system such as intermittent crashes, different attacks or recurrence of attacks which may be costly for the user.

If attacker gains access to system by changing the control flow of vulnerable program, it is known as buffer overflow attack. There are various ways performed by attacker in buffer overflow attack form which mostly used is to overflow the stack and modify the address or previously saved frame pointer. Due to high availability of requirements from service oriented platforms it is difficult to terminate an application. Patching the vulnerabilities is the better solution to avoid explosion of memory errors. Testing prototype with seven network daemon programs with known vulnerabilities show that the automatically generated patches can successfully fix the vulnerability. In this paper, we introduced system as SafeStack which automatically generates patches to stack-based buffer overflow vulnerabilities and apply the patches without stopping the vulnerable service. Memory access virtualization is the key technique of SafeStack that relocates memory objects to their desired locations. As attack is detected by system, Safestack identifies the stack objects that leads to stack buffer overflow, generates runtime patches which consist of vulnerability signatures, bug-triggering buffers and vulnerability treatments, and applies them to move these stack objects into protected memory areas. Hence we can say "SafeStack is highly efficient".

SafeStack consists of an online production system and an offline triage system to leverage failure diagnosis and availability of system. Specifically, the job of online system is to detect faults and apply patches to buffer whereas offline system identifies the bug-triggering buffers which then generates patches.

Therefore, the failure diagnosis in an application does not influence the work of applications. In our experiments, for each of nine stack-based buffer overflow bugs in eight applications, SafeStack can automatically generate patches just in a few seconds and enable the applications to survive from the subsequent attacks.

A DUAL SYSTEM ARCHITECTURE

Main goal is to preserve the availability by reducing the impact on the running application, a dual system architecture is used that consists of two systems i.e. an online production system and an offline triage system. The production environment is available to the application by online production system. To identify bug triggering buffers and generating patches offline triage system is used by application. The online production system detects exploits and also identifies its runtime types, in the triage system to exploit diagnosis the logger components logs into network inputs and the task of receiving patches and managing them is performed by patch management component and then these patches are applied with dynamic instrumentation tool by a runtime patch applicator. Extracting information about stack buffers from program binaries is performed by component named buffer information extractor. In shadow application bug-triggering buffers are detected by diagnosis engine component. According to results generated by diagnosis engine runtime patches are generated by patch generator and then to evaluate these patches i.e to find out whether these patches can resist the exploits is in offline triage system. Exploit report is created by patch evaluation component and patch management component which is used by programmers to find causes of exploit.

IMPLEMENTATION

SafeStack is a system that can automatically diagnose and patch stack-based buffer overflow vulnerabilities. The key technique of Safestack is virtualize memory accesses and move the vulnerable buffer into protected memory regions, which provides a fundamental and effective protection against recurrence of the same attack without stopping normal system execution. We developed a prototype on a system, and conducted extensive experiments to evaluate the effectiveness and performance of the system using a range of applications in real world attacks. We develop a technique, memory access virtualization, which allows runtime relocation of memory objects through binary instrumentation.

Memory Access Virtualization

The key technique of SafeStack is memory access virtualization, which allows memory objects to be relocated at runtime. It is the basis of both attack diagnosis and vulnerability patching. Memory access virtualization maps the original memory address of an object to a new address. Otherwise, the mapping has to be done carefully as the system needs to consider whether it is an out-of-bound access or a legitimate access to a different variable.

The offset from the stack pointer is determined at the compilation time. SafeStack replaces this original address with its corresponding new address before executing the instruction. The effective memory address is the sum of the starting address of the array and the size of the array element multiplied by the array index, and the addressing mode is “Base plus Index plus Offset.” The base is the base register, the offset is the value between the starting address of the array and the base register, and the index is the index register. SafeStack first calculates the starting address of the array, finds

the corresponding new address, calculates the final new address, and replaces the original address.

Stack Buffer Relocation

To relocate a stack buffer to the database table, SafeStack allocates the same size of the corresponding buffer on the other database table. Moves the content of stack from original database to new database table.

Bug-Triggering Buffer

It records bugs that have been triggered. It helps tracking and protecting same set of function malfunctioning when attacked by attacker.

Patch Generation and Evaluation

Stack buffer is isolated by generating patches when bug triggering stack is buffer is encountered. It is noticed that after patch is generated at runtime it includes vulnerability signature, its treatment and a bug-triggering buffer. As the bug-triggering buffer is examined in system, the bug signature is also seen on that time. The bug-triggering buffer consist of the buffer size and the offset. Mostly all vulnerability treatments are of the similar in the current implementation, i.e. divided in two set of memories. First it moves the stack buffer in memory and second it maps the memory. Patches are then forwarded to online production but before that these patches are evaluated. These patches are now applied on program by the system and then checks whether application can survive the exploit by replaying the input. If it fails, system rechecks patched application to encounter vulnerabilities. After failure of all patches generated by system, application still exploits then its not just caused only by stack buffer overflow and now system is ready to throw exception to production system. During this process a bug report is generated.

Runtime Patch Application

When receiving patches from the triage system, the patch management component in the production system notifies the runtime patch applicator to apply patches. For the patched application, once the vulnerability signature is matched, the memory access virtualization mechanism will move the bug-triggering buffers. The moved objects are randomly placed on the database to avoid them being overrun from each other

CONCLUSIONS

System SafeStack can automatically patch stack-based buffer overflow vulnerabilities. The system focuses on the vulnerability triggering stack buffers. SafeStack identifies the bug-triggering stack buffers and move them out of the stack to a protected space using memory access virtualization technique, the procedure continues by diagnosing vulnerabilities and finally generates runtime patches to “fix” the vulnerabilities temporarily to protect the applications from the subsequent recurrence of the same attacks. Prototype system has been developed and evaluated the system’s effectiveness using a range of applications with different bugs. The results demonstrate that our system can quickly generate runtime patches to successfully tolerate recurrence of the same attack.

REFERENCES

1. Beyond Security’s SecuriTeam. <http://www.securiteam.com>.
2. D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automatic generation of vulnerability-based signatures. In Proc. Of IEEE Symposium on Security and Privacy, 2006.

3. Bugtraq. Blackmoon ftp server buffer overflow vulnerability. <http://www.securityfocus.com/bid/3884/info>.
4. Bugtraq. Bugtraq mailing list. <http://www.securityfocus.com>.
5. T-C. Chiueh and F-H. Hsu. RAD: A compile-time solution to buffer overflow attacks. In Proc. of 21st Intl. Conf. on Distributed Computing Systems, 2001.
6. M. Costa, J. Crowcroft, M. Castro, L. Zhou A. Rowstron, L. Zhang, and P. Barham. Vigilante: End-to-End Containment of Internet Worms. In Proceedings of ACM Symposium on Operating Systems Principles, 2005.
7. Determina. How the memory firewall works. <http://www.determina.com/whitepapers/index.html>.
8. H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In Proceedings of USENIX Security Symposium, 2004.
9. C. Kreibich and J. Crowcroft. Honeycomb — creating intrusion detection signatures using honeypots. In Proc. of the Second Workshop on Hot Topics in Networks (Hotnets II), 2003.
10. L-C. Lam and T-C. Chiueh. Automatic extraction of accurate application-specific sandboxing policy. In Proc. of Seventh International Symposium on Recent Advances in Intrusion Detection, 2004.
11. “US-CERT Vulnerability Notes Database,” <http://www.kb.cert.org/vuls/bymetric?open\&start=1\&count=20>, 2013.
12. “Internet Security Threat Report,” <http://www.symantec.com/enterprise/threatreport/index.jsp>, 2013.
13. M. Nicholls, “Tutorial: SEH Based Exploits and the Development Process,” <http://www.ethicalhacker.net/content/view/309/2/>, 2013.
14. C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks,” Proc. Seventh Conf. USENIX Security Symp, pp. 63-78, 1998.
15. “A Stack Smashing Technique Protection Tool for Linux,” <http://www.anglefire.com/sk/stackshield>, 2012.
16. M. Prasad and T. Chiueh, “A Binary Rewriting Defense against Stack Based Buffer Overflow Attacks,” Proc. USENIX Annu. Technical Conf, pp. 211-224, 2003.
17. T. Chiueh and F. Hsu, “RAD: A Compile-time Solution to Buffer Overflow Attacks,” Proc. 21st Int’l Conf. Distributed Computing Systems, pp. 409-417, 2001.
18. O. Ruwase and M. Lam, “A Practical Dynamic Buffer Overflow Detector,” Proc. 11th Annu. Network Distributed System Security Symp, pp. 159-169, 2004.
19. C. Cowan, S. Beattie, J. Johansen, and P. Wagle, “PointGuard: Protecting Pointers from Buffer Overflow Vulnerabilities,” Proc. 12th Conf. USENIX Security Symp, pp. 91-104, 2003.
20. PaX Team, “PaX,” <http://pax.grsecurity.net>, 2013.
21. SafeStack: Automatically Patching StackBased Buffer Overflow Vulnerabilities Gang Chen, Hai Jin, Senior Member, IEEE, DeqingZou, Bing Bing Zhou, Zhenkai Liang, WeideZheng, and Xuanhua Shi